2. Beadandó feladat dokumentáció

Készítette:
Orosz Katalin
E-mail: zi5vc4@inf.elte.hu

Feladat:

Készítsünk programot, amellyel a következő játékot játszhatjuk. Adott egy $n \times n$ mezőből álló játékpálya, amelyeken falak, illetve járható mezők helyezkednek el, valamint ellenfelek járőröznek. A játékos célja, hogy ellenfeleit bombák segítségével minél gyorsabban legyőzze.

Az ellenfelek adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy folyamatosan előre haladnak egészen addig, amíg falba nem ütköznek. Ekkor véletlenszerűen választanak egy új irányt, és arra haladnak tovább.

A játékos figurája kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, de ha találkozik (egy pozíciót foglal el) valamely ellenféllel, akkor meghal.

A játékos bombát rakhat le az aktuális pozíciójára, amely rövid időn belül robban megsemmisítve a 3 sugáron belül (azaz egy 7 × 7-es négyzetben) található ellenfeleket (falon át is), illetve magát a játékost is, ha nem menekül onnan.

A pályák méretét, illetve felépítését (falak, ellenfelek kezdőpozíciója) tároljuk fájlban. A program legalább 3 különböző méretű pályát tartalmazzon.

A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a felrobbantott ellenfelek számát.

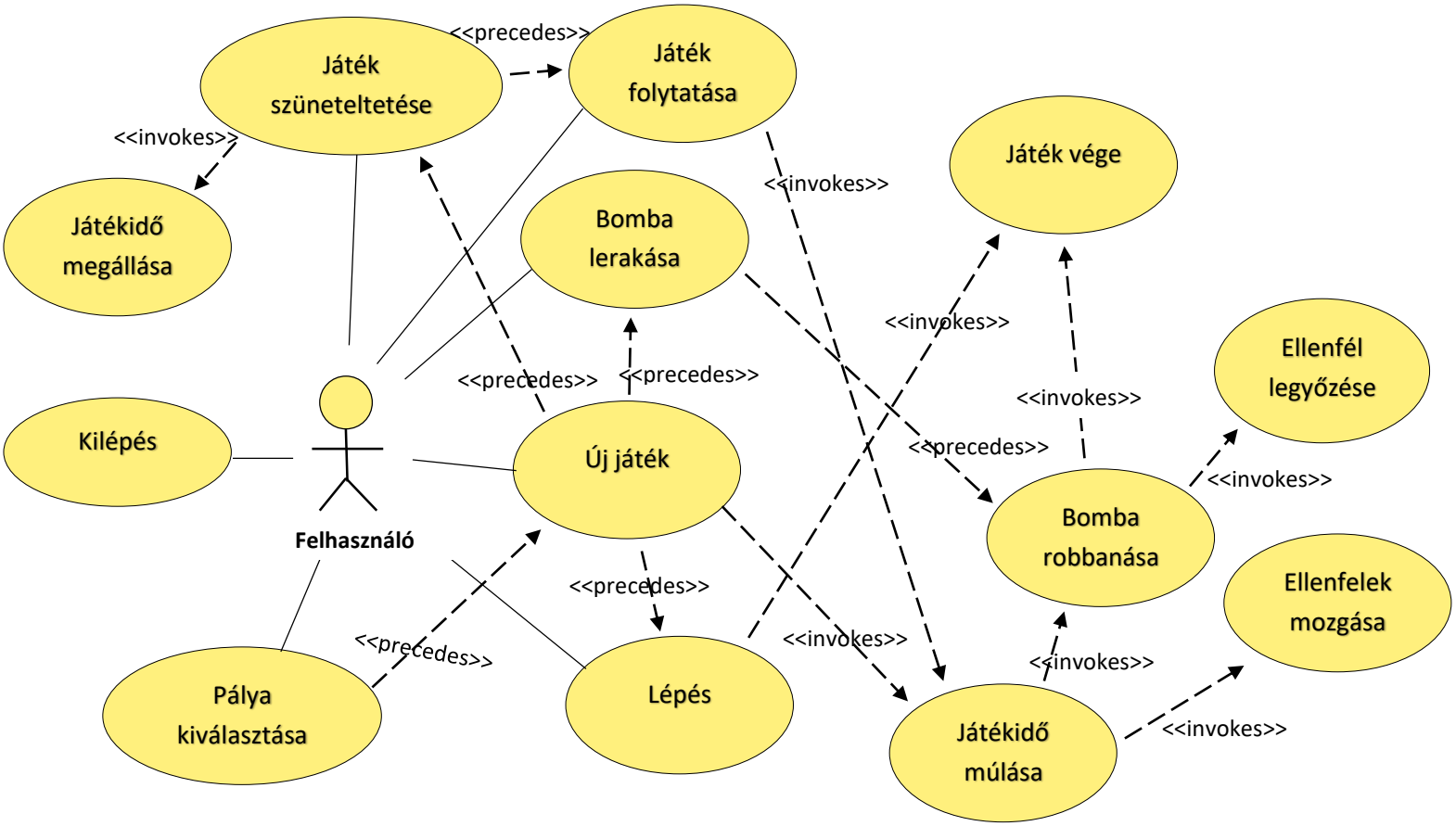
Elemzés:

- A játékban legalább 3 különböző nagyságú, előre fájlban előkészített pálya kiválasztására kell lehetőséget adni. Kiválasztás után új játékot lehet kezdeni az adott pályán.
- A feladatot egyablakos asztali alkalmazásként Windows Forms grafikus felülettel valósítjuk meg.
- Az ablakban elhelyezünk rádiógombokat, melyek különböző pályákra utalnak (látható lesz a méretük), és egy New Game gombot, mellyel új játékor indíthat a felhasználó.
- Az új játék kezdetekor a játék végeztéig megjelenítünk egy státuszsort, amely az eltelt időt jelzi, másodpercben, valamint a legyőzött ellenfelek számát. Ezen kívül még egy gombot is Stop (kattintás után Resume) felirattal, melyre kattintva a játékosnak lehetősége nyílik a játék szüneteltetésére, majd folytatására.

- A játék pályája címke (label) típusú kontrolokkal lesz megvalósítva, ezeket az alkalmazás tárolja egy kétdimenziós tömbben. Vizuálisan fogják reprezentálni a különböző mezőket: fal (az egészet feketére festjük), üres (üres tartalom), ellenfél, játékos, bomba (unikód szimbólum tartalomként). A felhasználó a billentyűzet nyilaival mozgathatja majd a játékost a pálya keretein belül, a falakon nem mehet át, illetve a "B" lenyomásával bombákat tehet le, melyek 3 másodperc múlva felrobbannak 7x7-es környezetben.

- A játék automatikusan feldob egy dialógusablakot, amikor vége a játéknak (felrobbantottuk az összes ellenfelet, vagy felrobbantunk mi magunk, vagy találkoztunk egy ellenféllel).

- A felhasználói esetek az 1. ábrán láthatóak.



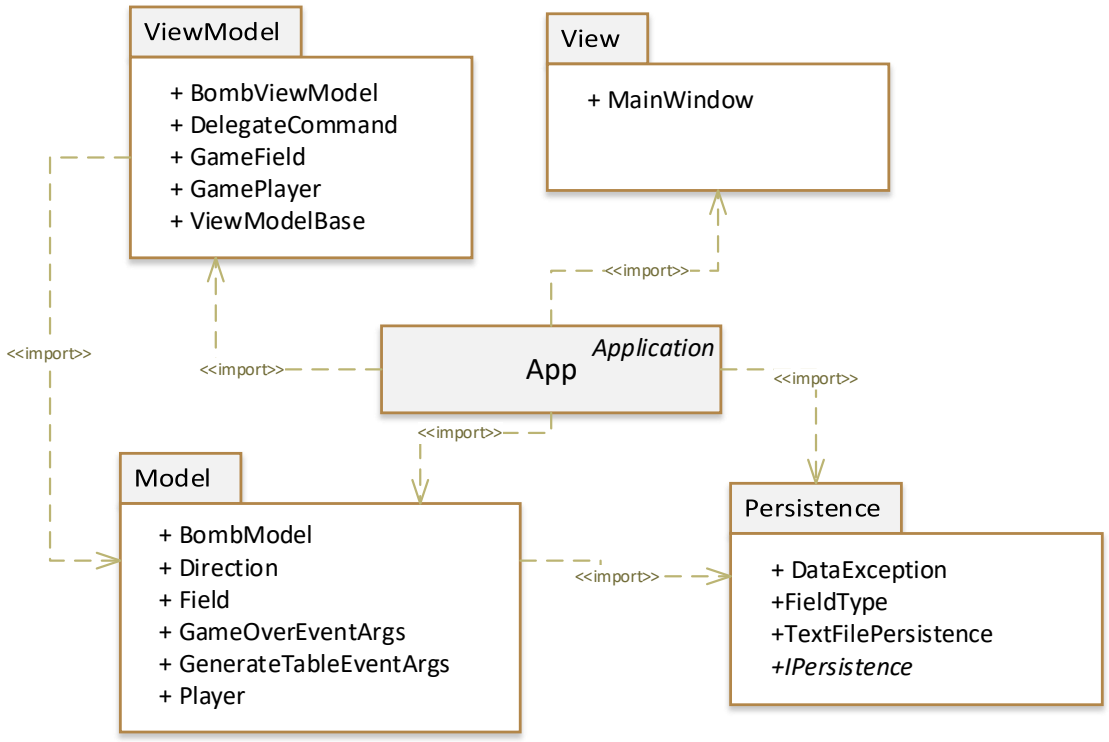
1. ábra: Felhasználói esetek

Tervezés

- **Programszerkezet**:
  - A programot MVVM architektúrában valósítjuk meg, ennek megfelelően **`View`**, **`Model`**, **`ViewModel`** és **`Persistence`** névtereket valósítunk meg az alkalmazáson

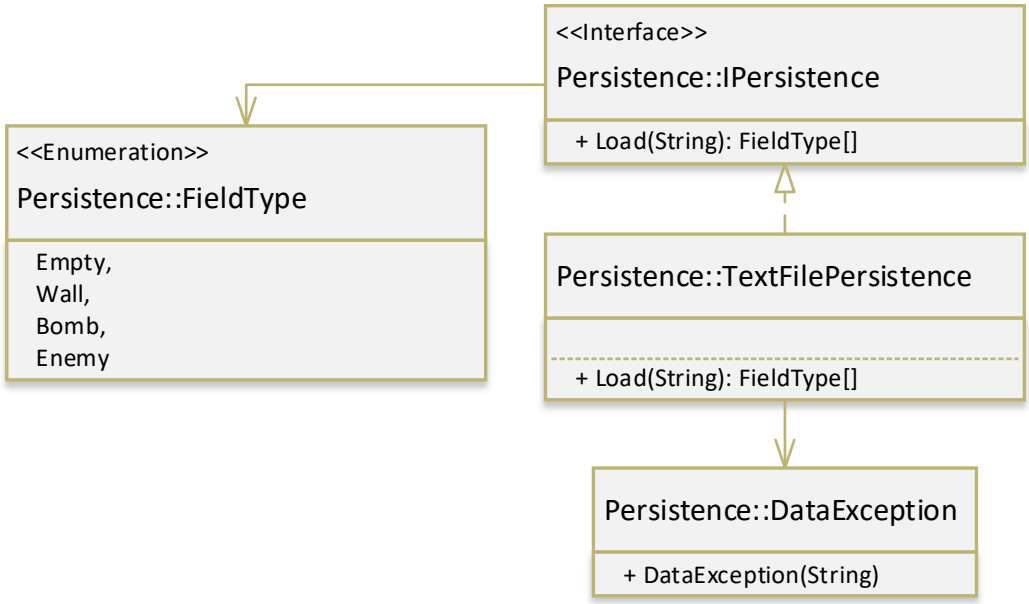
belül. A program környezetét az alkalmazás osztály (**App**) végzi, amely példányosítja a modellt, a nézetmodellt és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést. A program csomagszerkezete a 2. ábrán látható.

- A program szerkezetét két projektre osztjuk implementációs megfontolásból: a **Persistence** és **Model** csomagok a program felületfüggetlen projektjében, míg a **ViewModel** és **View** csomagok a WPF függő projektjében kapnak helyet.



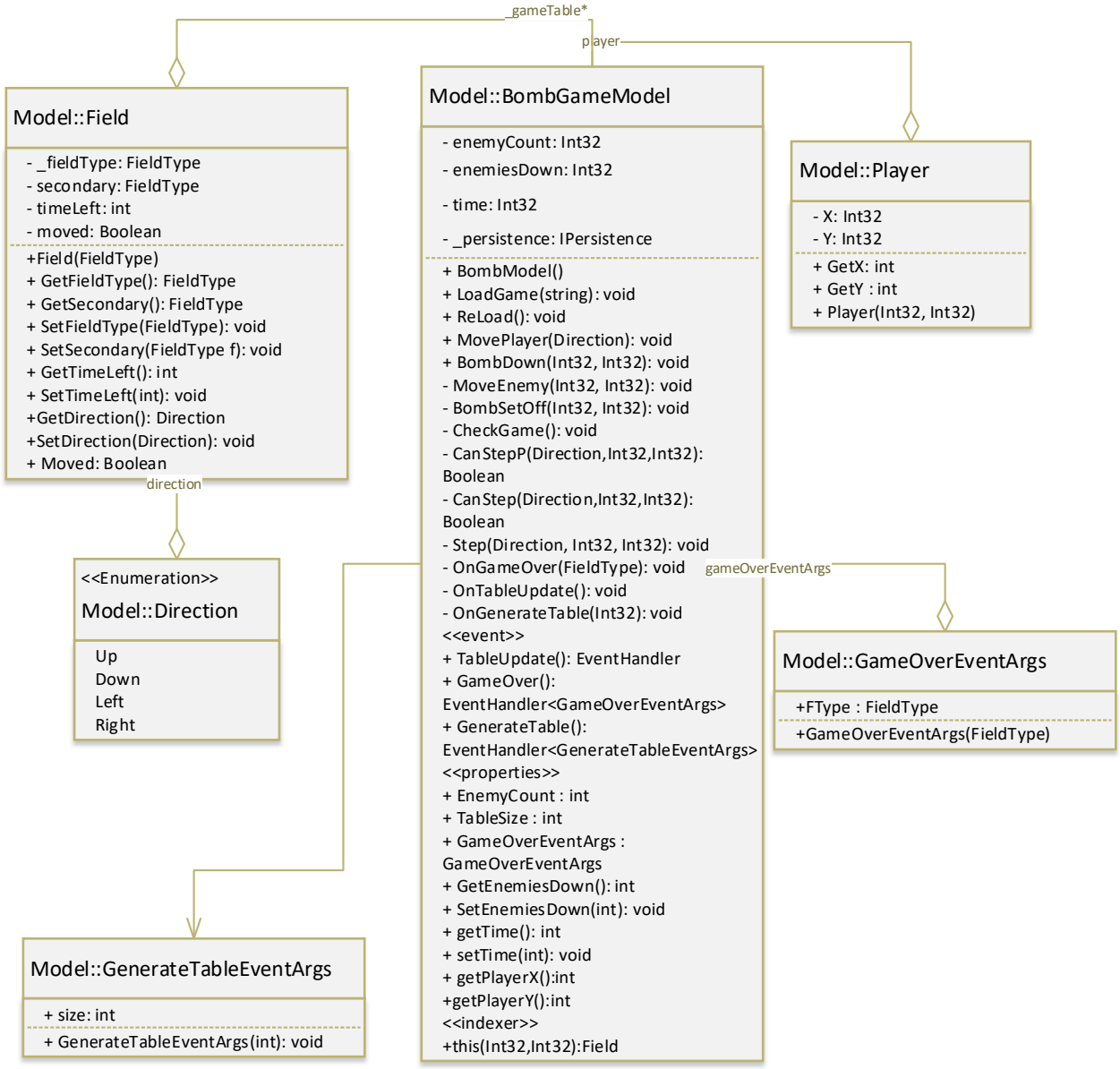
2. ábra: Az alkalmazás csomagdiagramja

- **Perzisztencia** (3. ábra):
  - Az adatkezelés feladata a kezdeti pályával kapcsolatos információk betöltése.
  - A **FieldType** enumeráció egy mezőket azonosító típust biztosít, melyek fajtái: **Empty** az üres, **Wall** a fal, **Bomb** a bomba és **Enemy** az ellenfél mező.
  - A kezdeti adatok betöltésére lehetőséget ad az **Ipersistence** interfész (**Load**).
  - Az interfészt szöveges fájl alapú adatkezelésre a **TextFilePersistence** osztály valósítja meg. A fájlkezelés során fellépő hibákat a **DataException** kivétel jelzi.
  - A fájl izomorf leképezése a játéktáblának, N sor mindegyikében N számot tartalmaz szóközökkel elválasztva. A számok 0, 1 vagy 3 lehetnek, ahol 0 az üres mezőt, 1 a falat és 3 az ellenfelet reprezentálja.



3. ábra: A Persistence csomag osztálydiagramja

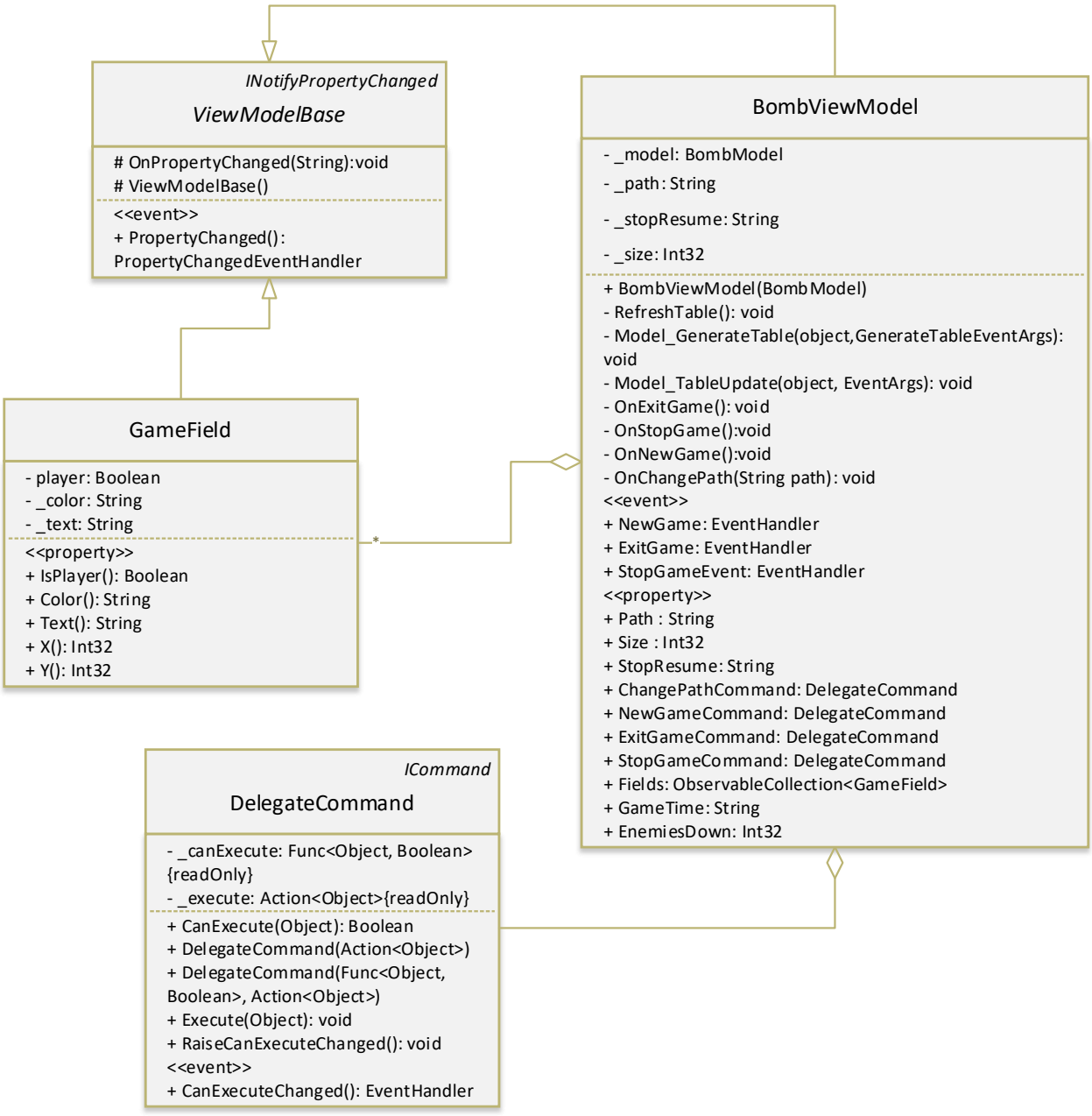
- **Modell** (4. ábra):
  - A modell lényegi részét a **`BombModel`** osztály valósítja meg, amely szabályozza a pálya tevékenységeit, valamint a játék egyéb paramétereit, úgymint az idő (**`time`**), a legyőzött ellenfelek (**`enemiesDown`**) és a kezdeti ellenfelek (**`enemyCount`**). A típus lehetőséget ad a játékos mozgatására (**`MovePlayer`**), és bomba lerakására (**`BombDown`**). Az idő előreléptetését időbeli lépések végzésével (**`ReLoad`**) tehetjük meg.
  - A modell példányosításkor megkapja az adatkezelés felületét, amelynek segítségével lehetőséget ad a betöltésre (**`LoadGame`**).
  - A modell három eseményen keresztül kommunikál a nézettel: **`GenerateTable`** (új pályát generál, kezdetben teljesen üreset, a **`GenerateTableEventArgs`** tartalmazza a pálya méretét), **`GameOver`** (vége van a játéknak, vagy az ellenfelek győztek, vagy a játékos a **`GameOverEventsArgs`** függvényében), **`TableUpdate`** (a tábla újratöltése).
- **Nézetmodell** (5. ábra):
  - A nézetmodell megvalósításához felhasználunk egy általános utasítás (**`DelegateCommand`**), valamint egy ős változásjelző (**`ViewModelBase`**) osztályt.
  - A nézetmodell feladatait a **`BombViewModel`** osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, a játékból való kilépéshez, a játék szüneteltetéséhez vagy folytatásához, valamint a betöltendő pálya kicseréléséhez. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását **`(_model`**), de csupán információkat kér le tőle. Direkt nem avatkozik a játék futtatásába.



4. ábra: A Modell csomag osztálydiagramja

- A játékmező számára egy külön mezőt biztosítunk (**`GameField`**), amely eltárolja a pozíciót, a szöveget (icon), a színt, valamint hogy rajta van-e a játékos a mezőn. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (**`Fields`**).

- **Nézet:**
  - A nézet csak egy képernyőt tartalmaz, a **`MainWindow`** osztályt. A nézet egy rácsban tárolja a játékmezőt, a lehetséges pályaméretek kiválasztását a játékkezdő gombbal együtt, a kilépés gombot, a státuszsort, valamint a szüneteltető gombot. A játékmező egy **`ItemsControl`** vezérlő, ahol dinamikusan felépítünk egy rácsot (**`UniformGrid`**), amely



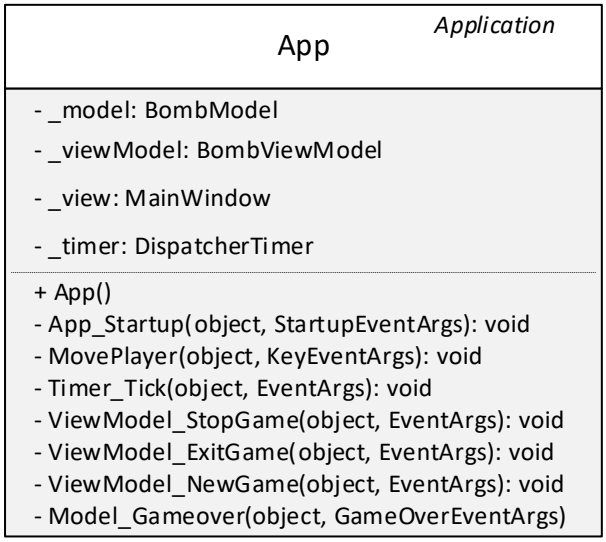
5. ábra: A ViewModell csomag osztálydiagramja

címkékből (label) áll. Minden adatot adatkötéssel kapcsoltunk a felülethez, továbbá azon keresztül szabályozzuk a label által megjelenített ikonokat is (mint text attribútum).

- A játék végét jelző üzenetek megjelenítését beépített dialógusablakok segítségével végezzük.

- **Környezet** (6. ábra)**:**
  - Az **`App`** osztály feladata az egyes rétegek példányosítása (**`App_Startup`**), összekötése, a nézetmodell, valamint a modell eseményeinek lekezelése, és ezáltal a játék, az adatkezelés, valamint a nézetek szabályozása.

- A játék léptetéséhez tárol egy időzítőt is (**`_timer`**), amelynek elindítását vagy leállítását szabályozza egyes funkciók hatására.



6. ábra: A vezérlés osztálydiagramja

Tesztelés:

- A modell funcionalitása egységtesztek segítségével lett ellenőrizve a **`BombGameTest`** osztályban.
- Az alábbi tesztesetek kerültek megvalósításra:
  - **`BombGameModelLoadGameTest`**: Új játék indítása, a mezők betöltésének tesztelése mockolt perzisztencia réteggel, valamint a kezdeti adatok beállítása (ellenfelek száma, játékos elhelyezése).
  - **`BombGameModelMovePlayerTest`**: Játékos lépései hatásainak ellenőrzése, esemény kiváltásának ellenőrzése.
  - **`BombGameModelReLoadTest`**: A játékbeli idő kezelésének ellenőrzése, ellenfelek mozgatásának helyessége. Játék vége esemény ellenőrzése, random lefutásra is.
  - **`BombGameModelBombDownGameTest`**: Bombák helyes lerakása, bombák felrobbanásának következményei.